A Software Simulation of Transition P Systems in Haskell

Fernando Arroyo\(^1\), Carmen Luengo\(^1\), Angel V. Baranda\(^2\), and Luis de Mingo\(^3\)

\(^1\) Dept. Lenguajes, Proyectos y Sistemas Informáticos
Escuela de Informática, Universidad Politécnica de Madrid
Ctra. de Valencia km. 7, 28031 Madrid, Spain
{farroyo,cluengo}@eui.upm.es
http://www.lpsi.eui.upm.es

\(^2\) Dept. Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid
Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain
http://www.dia.fi.upm.es

\(^3\) Dept. Organización y Estructura de la Información
Escuela Universitaria de Informática, Universidad Politécnica de Madrid
Ctra. de Valencia Km. 7, 28031 Madrid, Spain
lfmingo@eui.upm.es
http://www.oei.eui.upm.es

Abstract. P systems are a parallel and distributed computational model, based on the membrane structure notion. Membranes define regions. Inside regions, objects and rules are placed in order to make evolve the P system. Evolution is achieved by transitions between two consecutive system configurations. Therefore, a computation can be obtained as a transitions series between consecutive configurations. Where and how P systems can be implemented is nowadays an open problem, but implementation on digital computers could be one way to show the capabilities of such systems. This paper presents a transition P systems implementation in Haskell, based on a theoretical framework previously developed.

1 Introduction

Transition P systems were introduced by G. Păun \(^5\). They are the simplest variant of P systems, however, they have the essential components of many variants of P systems. In fact, in any variant of P systems two basic components can be found: the static structure and the dynamic structure. The static structure consists of the membrane structure and the multisets or strings associated with each region defined by the membrane structure, while the dynamic structure consists of the rules associated with the regions defined by the membrane structure of the P system. These rules are responsible for the P system evolution, making changes in the static structure, changing the multisets of strings associated with

\(*\) Work partially supported by contribution of EU commission under The Fifth Framework Programme, project “MolCoNet” IST-2001-32008.

© Springer-Verlag Berlin Heidelberg 2003
each region, or even changing the membrane structure. Rules can be defined in many different ways, depending on the variant of P system considered.

One of the main goals of the implementation described here is to keep close enough the software to the philosophy of membrane computing. We try to achieve this, keeping in the different software modules the fundamental functionality of the corresponding ingredients of P systems. Therefore, any item (multiset, rule, region, membrane, etc) has its corresponding module in the developed software. Moreover, the functionality that the module provides to the software system is what it is expected from it. The static and dynamic structures of P systems have also their corresponding projection on the software.

The following sections are devoted to explain the software architecture in connection with a previous theoretical work of formalization corresponding to transition P systems. The formalization work has been essential in order to analyze and design the software architecture.

2 System Architecture Description

The current implementation of transition P system has been written in Haskell. The language has been chosen in order to keep as close as possible the implementation to a previously developed theoretical work, see [4],[3],[1] and [2]. Here we describe the software architecture of the implementation, giving a short description of the main software modules (implementing Abstract Data Types) and their functional dependencies.

2.1 Abstract Data Type Multiset

This module, ADTMultiset, implement the multiset concept. The module provides the Haskell type Multiset defined as follows:

- data Multiset a = CMultiset [(a, Int)]

From the above data definition in Haskell, multisets have been defined as lists of tuples with two elements: the first one is a char and the second one is the number of copies of the first element in the multiset.

This module also provides the most commonly used function in transition P system over multisets. The following list shows the type of such functions in Haskell:

- Empty multiset:
  - isEmptyMS :: Multiset a → Bool
- Multiset Union
  - (∧::+::∥) :: Eq a => Multiset a → Multiset a → Multiset a
- Multiset Intersection
  - (∧::\::∥) :: Eq a => Multiset a → Multiset a → Multiset a
- Multiset Difference
  - (∧::\::∥) :: Eq a => Multiset a → Multiset a → Multiset a